CS 2130 Final Exam

Name _____

You MUST write your e-mail ID on **EACH** page and put your name on the top of this page, too.

If you are still writing when "pens down" is called, your exam will not be graded – even if you are still writing the honor pledge. So please do that first. Sorry to have to be strict on this!

There are 16 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points, so be sure to look over all the questions and plan your time accordingly.

This exam is CLOSED text book, closed-notes, closed-cell phone, closed-smart watch, closed-computer, closed-neighbor, etc. You may **not** discuss this exam with anyone until after all the exam times have ended. Please write and sign the honor pledge below.

_____

_____

_____

_____

_____

**** stack smashing detected ***: terminated*
Aborted (core dumped)

Page 2: C to ISA

Throughout the semester, we took a bottom-up approach, building from electricity on wires to C. The next few questions take a top down approach to Computer Systems and Organization.

Consider the following C function to sum the elements in an array.

```
int sumArray(int *x, size_t n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += _____________;
    }
    return sum;
}
```

1. [4 points] Which of the following could be used in the blank above to correctly calculate the sum of all elements in the array? (Fill in the circle completely for all that apply)

- ◯ `x[i]`
- ◯ `x.i`
- ◯ `x->i`
- ◯ `x + i`
- ◯ `x + 4*i`
- ◯ `*(x + i)`
- ◯ `*(x + 4*i)`
- ◯ `&(x[0]) + i`

Page 3: C to ISA, continued

2. [10 points] Rearrange the following assembly instructions so that they implement `sumArray` function in x86-64 assembly. Write the number corresponding to each instruction on the lines provided to the right; you do not need to rewrite the entire instruction. Some order has been provided for you. *While all blanks will be used, not all instructions will be used.*

1. `addl 4(%rcx,%rdi), %eax`
2. `addl 4(%rdi,%rcx), %eax`
3. `addl (%rcx,%rdi,4), %eax`
4. `addl (%rdi,%rcx,4), %eax`
5. `addq $1, %rcx`
6. `addq $1, %rdi`
7. `addq $1, %rsi`
8. `cmpq %rcx, %rsi`
9. `je label2`
10. `jne label1`
11. `label1:`
12. `label2:`
13. `retq`
14. `sumArray:`
15. `testq %rsi, %rsi`
16. `xorl %eax, %eax`
17. `xorl %ecx, %ecx`
18. `xorl %edi, %edi`

14 `sumArray:`

___

___

___

___

11 `label1:`

___

___

___

___

13 `retq`

Page 4: C to ISA, continued

3. [10 points] Implement `sumArray` using our Toy ISA. Part of the code has been provided for you; fill in the missing instructions. In this implementation, the array is stored at memory address `0x50`, storing only 1-byte values; the program ends by saving the sum to memory at address `0x99`; and the program is loaded at memory address `0x00`. *Our Toy ISA is described on page 16*.

```
r0 = 0x09 // n = 9
r1 = 0x50 // address of array
r2 = 0x00 // result
r3 = 0x15
if r0 <= 0, set pc = r3
// Complete the following instructions (scratch space)

// Update loop condition



// Read from array and add



// Increment counter(s)



// Check condition and jump
r3 = 0x0C
if r0 <= 0, set pc = r3
r3 = 99
M[r3] = r2 // store result
halt
```

Write your program in binary below. Only the final program below will be graded for this question. If you use fewer instructions, use no-ops to fill in the rest. We have filled in some of the instructions and immediate values already.

60	09	64	50	68	00	6C	15					
00	01	02	03	04	05	06	07	08	09	0a	0b	0c

						0C			99		80	00
0d	0e	0f	10	11	12	13	14	15	16	17	18	19

Page 5: Binary

4. [3 points] Convert the 6-bit two's complement binary number `100111` into decimal.

Answer

5. [3 points] What is `0x8675E094 & 0x12345678`? Answer in hexadecimal.

Answer

6. [3 points] Which of the following C expressions are equivalent to `x | y`, where `x` and `y` are declared as ints? (Fill in the circle completely for all that apply)

- ◯ `~x & ~y`
- ◯ `~x ^ ~y`
- ◯ `!(!x & !y)`
- ◯ `~(~x & ~y)`
- ◯ `!(!x ^ !y)`
- ◯ `~(x + y)`

Page 6: Circuits

7. [8 points] In the space below, draw a circuit with three 1-bit inputs and one 1-bit output. Label the inputs **x**, **y**, and **s**. Construct the circuit using only wires and gates from the set {and, or, not, xor}. If `s` is 1, output `x ^ y`; if `s` is 0, output `x`. That is, draw a circuit of gates that does the same as `(s ? x ^ y : x)`. *Hint: writing out a truth table can help you design and/or check your circuit.*

Page 7: C

8. [3 points] Consider the following declarations and initialization of variables named `x`. For which would C consider `x` to be true? (Fill in the circle completely for all that apply)

   ◯ `long x = 10;`

   ◯ `int x = 0;`

   ◯ `double x = 0.1;`

   ◯ `char x = '\0';`

   ◯ `char x = '0';`

   ◯ `char *x = NULL;`

9. [3 points] Select the statement that will correctly print the first two command line arguments to stdout. For example, the correct output for

   ```
   ./a.out 1 two three five
   ```

   is: "1 two". (Fill in the circle completely)

   ◯ `printf(argv[0], argv[1]);`

   ◯ `printf(argv[1], argv[2]);`

   ◯ `printf("%s %s\n", argv[0], argv[1]);`

   ◯ `printf("%d %d\n", argv[0], argv[1]);`

   ◯ `printf("%s %s\n", argv[1], argv[2]);`

   ◯ `printf("%d %d\n", argv[1], argv[2]);`

10. [3 points] In this course, we primarily used the CS department portal to compile and run our code. Assuming that you have written code in a file named `csofinal.c`, write the command to compile your code with 2 levels of optimization and debugging information.

    `mst3k@portal03:~$` _____

Page 8: Memory

Consider the following code, shown with line numbers which are not part of the code itself. The code contains one or more memory errors. Use this code for Questions 11-13.

```
1.  // Determine if a number is odd
2.  int isOdd(int *x) {
3.      return (*x) % 2;
4.  }
5.
6.  // Sum up to the first n even numbers
7.  int sumFirstEvens(int *array, int n) {
8.      int *cpy = (int *)malloc(sizeof(n));
9.      int *sum = (int *)malloc(sizeof(int));
10.     int *cpy2 = cpy;
11.     int *sum2 = sum;
12.     for (int i = 0; i < n; i++)
13.         cpy[i] = array[i];
14.     while (!isOdd(cpy)) {
15.         *sum += *cpy;
16.         cpy += 1;
17.     }
18.     free(sum);
19.     return *sum2;
20. }
```

11. [3 points] The code above has one memory leak. After which line should we add a `free`? For example, if we should free after `return (*x) % 2;`, write 3.

Answer

12. [3 points] What should be freed? That is, what should we pass to the call `free();`?

Answer

Page 9: Memory and C

13. [6 points] For each of the following memory errors, give the line where the error occurs in the code on the previous page. If the error does not occur, put an X in on the line.

    ____ Accesses uninitialized memory

    ____ Accidentally casts to a pointer

    ____ Possible stack buffer overflow

    ____ Possible heap buffer overflow

    ____ Uses after free

    ____ Fails to use `sizeof` or uses `sizeof` incorrectly

14. [3 points] Assume the struct `examInfo` is defined as follows. What is `sizeof(examInfo[10])`? *See the assumptions on page 16 to compute an exact number.*

```
typedef struct {
    long minutesAllowed;
    int  maxScore;
    int  score;
    char *studentName;
} examInfo;
```

Answer

15. [3 points] Remember that `printf`, `puts`, and other functions invoke `write` to do the actual output to a file, socket, or the terminal. How does the `write` function output? (Fill in the circle completely)

    ◯ Setting specific bytes of memory that represent the screen or file contents

    ◯ Calling (with `call` assembly instruction) a function stored in user memory

    ◯ Calling (with `call` assembly instruction) a function stored in kernel memory

    ◯ Using the special `syscall` assembly instruction

Page 10: Writing C

16. [12 points] Implement the `strnsep` function, included in the manual page on this exam. You may **not** use `strsep` or any library functions (except `strlen()`).

```
char *strnsep(char **stringp, const char *delim, size_t n) {

}
```

Page 11: More Questions

17. [4 points] Answer the following True/False questions by writing either T or F in the blank provided.

    ____ Each process has its own view of all of memory.

    ____ The lower addresses (close to 0) are reserved for kernel memory.

    ____ Garbage collectors (like in Java) are able to free all garbage in memory.

    ____ When accessing struct `baz` through a pointer `p`, the following are both valid methods for accessing field `d`: `p->d` and `(*p).d`

18. [3 points] Suppose you found a security vulnerability in code that you did not write. When should you publish the vulnerability to a public repository of vulnerabilities (such as CVE)?

    ◯ Immediately upon discovering it

    ◯ Immediately after reporting it to the author of the software

    ◯ After providing the author of the software enough time to fix it, whether or not it was fixed

    ◯ Only after it is fixed

19. [3 points] Which of the following should not be included in a header (.h) file? That is, which should we instead put in our .c files? (Fill in the circle completely for all that apply)

    ◯ `unsigned long answer = 42;`

    ◯ `struct foo {int x; double d;};`

    ◯ `int rand() { return 4; }`

    ◯ `#define MAX_LEN 42`

    ◯ `void *smartAllocate(size_t numInts);`

    ◯ `typedef struct {long g; char *a} bar;`

Page 12: More Questions

For the next two questions, we want to read and print one character at a time to stdout using the following code:

```
void readAndPrintChar() {
    char c = '\0';
    _____________1_____________ {
        ___________2_________________;
    }
}
```

20. [3 points] Which of the following could we use in the first blank to correctly loop and read one character at a time (buffered or unbuffered)? (Fill in the circle completely for all that apply)

- ◯ `while(read(0, char, 1) == 1)`
- ◯ `while(read(0, &char, 1) == 1)`
- ◯ `while((c = fgetc(stdin)) != EOF)`
- ◯ `while(fgets(&c, 1, stdin))`

21. [3 points] Which of the following could we use in the second blank to correctly print one character? (Fill in the circle completely for all that apply)

- ◯ `write(1, &c, 1);`
- ◯ `printf("%s", c);`
- ◯ `printf("%s", &c);`
- ◯ `printf("%c", c);`

Page 13: Free Points

22. [2 points] **(Free Points)** We covered a lot of material in the course this semester, from electricity on wires through most of C. What is one topic you wished we had spent more time discussing in class or lab?

23. [2 points] **(Free Points)** My father was 86 years old when he passed this semester, and he was never great with computers. Explain your favorite part of the course material to him in a way that he could appreciate and understand. (Only serious answers will receive extra credit.)

Page 14: Man Page

STRSEP(3) Linux Programmer's Manual STRSEP(3)

NAME

`strsep, strnsep` - extract token from string

SYNOPSIS

```
#include <cso1final.h>

char *strsep(char **stringp, const char *delim);

char *strnsep(char **stringp, const char *delim, size_t n);
```

DESCRIPTION

If `*stringp` is `NULL`, the `strsep()` function returns `NULL` and does nothing else. Otherwise, this function finds the first token in the string `*stringp`, that is delimited by one of the bytes in the string delim. This token is terminated by overwriting the delimiter with a null byte (`'\0'`), and `*stringp` is updated to point past the token. In case no delimiter was found, the token is taken to be the entire string `*stringp`, and `*stringp` is made `NULL`.

The `strnsep()` function is similar, except in the case no delimiter was found in the first `n` bytes of `*stringp`. In this case, the token consists of the first `n-1` bytes. This token is terminated by overwriting the `n`th byte with a null byte (`'\0'`), and `*stringp` is updated to point past the token.

RETURN VALUE

The `strsep()` and `strnsep()` functions returns a pointer to the token, that is, they return the original value of `*stringp`.

Page 15: Man Page

FGETC(3) Linux Programmer's Manual FGETC(3)

NAME

`fgetc, fgets, getc, getchar, ungetc` - input of characters and strings

SYNOPSIS

```
#include <stdio.h>

int fgetc(FILE *stream);

char *fgets(char *s, int size, FILE *stream);

int getc(FILE *stream);

int getchar(void);
```

DESCRIPTION

`fgetc()` reads the next character from stream and returns it as an `unsigned char` cast to an `int`, or `EOF` on end of file or error.

`getc()` is equivalent to `fgetc()` except that it may be implemented as a macro which evaluates stream more than once.

`getchar()` is equivalent to `getc(stdin)`.

`fgets()` reads in at most one less than size characters from stream and stores them into the buffer pointed to by `s`. Reading stops after an `EOF` or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('`\0`') is stored after the last character in the buffer.

Calls to the functions described here can be mixed with each other and with calls to other input functions from the stdio library for the same input stream.

For nonlocking counterparts, see `unlocked_stdio(3)`.

RETURN VALUE

`fgetc()`, `getc()` and `getchar()` return the character read as an `unsigned char` cast to an `int` or `EOF` on end of file or error.

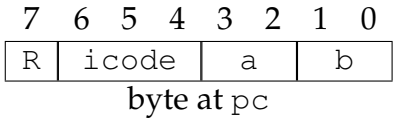
`fgets()` returns `s` on success, and `NULL` on error or when end of file occurs while no characters have been read.

Page 16: Assumptions and Our ISA

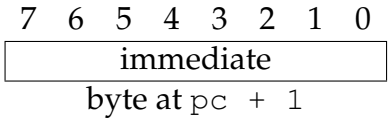
Assumptions. Assume unless otherwise specified:

- All necessary `#includes` have been used
- `char`, `short`, `int`, and `long` are 8-, 16-, 32-, and 64-bits long, respectively
- The compiler pads pointers where it is allowed to do so such that `sizeof(struct X)` is:
  - an even multiple of the size of its largest field
  - the smallest such multiple big enough to store all its fields

Our Example ISA. This is the ISA described in class and used in Lab 4 and Homework 3. Each instruction is one (or two) bytes, defined as:



byte at `pc`



byte at `pc + 1`

If the reserved bit (bit 7) in our instruction is 0, the following table defines our instruction encoding.

icode	b	behavior
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
6	0	`rA` = read from memory at `pc + 1`
	1	`rA` += read from memory at `pc + 1`
	2	`rA` &= read from memory at `pc + 1`
	3	`rA` = read from memory at the address stored at `pc + 1`
		For icode 6, increase `pc` by 2 at end of instruction
7		Compare `rA` as 8-bit two's-complement to `0`
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal